Санкт-Петербургский государственный университет

ГУДЗЕНКО Алина Артемовна

Выпускная квалификационная работа

«Разработка мобильного Android-приложения для организации мероприятий и встреч»

Уровень образования: бакалавриат
Направление *09.03.03 «Прикладная информатика»*
Основная образовательная программа СВ.5078
Профиль *«Прикладная информатика в области искусств и гуманитарных наук»*

Научный руководитель:
Канд. физ.-мат. наук, доцент, кафедра информационных систем в искусстве и гуманитарных науках, СПбГУ
Щербаков Павел Петрович

Консультанты:
Ст. преподаватель,
Кафедра информационных систем в искусстве и гуманитарных науках, СПБГУ
Мбого Ирина Анатольевна

Рецензент:
Общество с ограниченной Ответственностью «Резолла»
Гордеев Александр Владимирович, профессор, Кафедра вычислительных систем и сетей, «Санкт-Петербургский государственный университет аэрокосмического приборостроения»

Санкт-Петербург
2023

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ИСКУССТВ
Кафедра информационных систем в искусстве и гуманитарных науках
Основная образовательная программа
«Прикладная информатика в области искусств и гуманитарных наук»

УТВЕРЖДАЮ
Заведующий кафедрой

_____

ЗАДАНИЕ
по подготовке выпускной квалификационной работы студентки Гудзенко Алины Артемовной

1. Тема работы: Разработка мобильного Android-приложения для организации мероприятий и встреч
2. Срок сдачи студентом законченной работы: июнь 2023
3. Срок сдачи текста выпускной квалификационной работы для выкладывания в Blackboard: 22 мая 2023 г.
4. Исходные данные к работе: оригинальная идея.
5. План-график выполнения квалификационной работы:

Номера и содержание этапов работы	**Плановая дата сдачи**
1. Формирование требований к мобильному приложению.	октябрь
2. Проектирование архитектуры приложения.	декабрь
3. Подбор технологий для реализации.	март
4. Реализация мобильного приложения	апрель
5. Написание теоретической части диплома	апрель-май
6. Защита теоретической и практической частей диплома	июнь

Консультант по работе: Мбого Ирина Анатольевна
Руководитель от кафедры: Щербаков Павел Петрович

_____
__

(должность, Фамилия Имя Отчество, подпись)

Задание принял к исполнению_____ Подпись студента_____
_____ (дата)

АННОТАЦИЯ

выпускной квалификационной работы

Гудзенко Алины Артемовны

«Разработка мобильного Android-приложения для организации мероприятий и встреч»

Целью работы является разработка мобильного Android-приложения для организации мероприятий и встреч.

Данная выпускная квалификационная работа состоит из 4 частей: формирование требований к мобильному приложению и выбор подходящей архитектуры, проектирование архитектуры мобильного приложения, подбор технологий для реализации и реализация данного мобильного приложения. В первой части будут выделены исходные данные приложения и подобрана подходящая архитектура для данного приложения. Во второй части будет спроектирована выбранная архитектура. В третьей части будут подобраны подходящие для данного приложения технологии. В четвертой части описан процесс разработки приложения по модулям и функционалу.

Автор работы _____ _____

подпись (фамилия, имя, отчество)

Руководитель работы _____ _____

подпись (фамилия, имя, отчество)

Оглавление

Определения

Архитектура приложений — это структурный принцип, по которому создано приложение. Каждому архитектурному виду свойственны свои характеристики, свойства и отношения между компонентами.

Компонент — мелкая или крупная логическая и независимая часть архитектурной системы приложения.

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.

NoSQL — обозначает нереляционные типы баз данных, которые хранят данные в формате, отличном от реляционных таблиц.

Биндигн данных (data binding) — это общий метод, который связывает вместе два источника данных/информации и поддерживает синхронизацию данных.

API (Application Programming Interface) – что значит программный интерфейс приложения, это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов.

Интегрированная среда разработки (IDE) – это программное приложение, которое помогает программистам эффективно разрабатывать программный код.

XML – это язык разметки, который предоставляет правила для определения любых данных

Нативная разработка – это классическое решение, которое требует писать приложения под каждую платформу по отдельности, используя разные языки и учитывая особенности каждой платформы.

Кроссплатформенная разработка — это реализация приложения, которое работает на нескольких операционных системах.

Фреймворк (с англ. framework — «каркас, структура») — заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код.

Введение

В качестве ВКР мною было выбрано создание мобильного Android-приложения для организации мероприятий и встреч, в котором любой зарегистрированный пользователь сможет найти мероприятие по своим критериям или организовать свою встречу.

Целью данной работы, было разработка мобильного приложения для организации мероприятий и встреч. Данное приложение поможет пользователям найти или создать свое мероприятие по желаемым запросам. Зарегистрированные пользователи имеют возможность откликаться на мероприятия других пользователей. Все мероприятия расположены на карте и пользователь может выбрать увидеть ближайшие к нему мероприятия и посетить их. Также у зарегистрированных пользователей есть возможность создать свое мероприятие указав удобное для себя время и тематику мероприятия. Так же у пользователей есть возможность общаться с друг другом с помощью встроенного мессенджера.

Для достижения данной цели в рамках выполнения работы были поставлены следующие задачи:

1. Формирование требований к мобильному приложению и выбор подходящей архитектуры.
2. Проектирование архитектуры мобильного приложения.
3. Подбор технологий для реализации.
4. Реализация данного мобильного приложения.

Выполнение данных задач поможет в разработке данного приложения. Очень важно подобрать правильную архитектуру мобильного приложения, так как с ее помощью приложение будет работать стабильно и поможет при расширении функций приложения в дальнейшем. Так же очень важно

подобрать технологии для реализации, с помощью, которых приложение будет работать правильно и не потребуется больших усилий со стороны разработчиков для поддержки работы приложения.

1. Формирование требований к мобильному приложению и выбор подходящей архитектуры.

1.1 Исходные данные о приложении

Для того, чтобы сформулировать требования к архитектуре данного мобильного приложения и подобрать подходящую архитектуру необходимо проанализировать исходные данные мобильного приложения для организации мероприятий.

Мобильное Android-приложение для организации мероприятий и встреч предназначено для того что бы любой зарегистрированный пользователь мог открыть карту с метками, которые обозначают мероприятия, выбрать подходящее ему мероприятие на карте и откликнуться на него. Либо создать свое мероприятия и поставить метку на карту, для того, чтобы другие пользователи могли на него откликнуться.

Необходимый функционал приложения:

1. Возможность зарегистрироваться или авторизоваться в приложении.
2. Приложение должно подключатся к базе данных и загружать информацию о пользователе и существующих метках, а также чаты и сообщения пользователя.
3. Приложение должно отрисовывать метки с мероприятиями по заданным координатам.
4. У зарегистрированных пользователей должна быть возможность добавить мероприятие на карту со своими критериями и просмотреть информацию о мероприятиях других пользователей.
5. Возможность связаться с автором мероприятия и откликнуться на мероприятие.

Также, для формирования требований к архитектуре необходимо понимать какая команда будет разрабатывать данное приложение. На данный момент «Мобильное Android-приложение для организации мероприятий и встреч» будет разрабатываться в команде из одного разработчика, но в будущем планируется расширить команду. Поэтому необходимо подобрать такую архитектуру для приложения, что бы не возникло проблем при расширении команды разработчиков и было удобно разрабатывать в небольшой команде.

1.2Анализ требований

Исходя из всего вышесказанного можно сформулировать следующие требования, которым должна отвечать архитектура данного приложения:

1. Архитектура системы должна быть достаточно гибкой, т.е. должна допускать относительно простое, без коренных структурных изменений, развитие инфраструктуры и изменение конфигурации используемых средств, наращивание функций и ресурсов ИС в соответствии с расширением сфер и задач ее применения.
2. Должны быть обеспечены безопасность функционирования системы при различных видах угроз и надежная защита данных от ошибок проектирования, разрушения или потери информации, а также авторизация пользователей, управление рабочей загрузкой, резервированием данных и вычислительных ресурсов, максимально быстрым восстановлением.
3. Следует обеспечить комфортный, максимально упрощенный доступ разработчикам к управлению и результатам функционирования информационной системы.
4. Архитектура должна подходить команде разработчиков.

А также соответствовать следующим принципам:

- открытость – возможность системы допускать замену любого элемента системы без пересмотра системной архитектуры;
- модифицируемость – возможность изменения алгоритмов работы системы путем изменения конфигурационных данных;
- масштабируемость – возможность горизонтально и вертикально наращивать ресурсы системы с пропорциональным повышением производительности, таким образом, что при этом не возникает необходимости модернизации программного обеспечения системы или проведения структурных изменений системы;
- тестируемость – возможность установления факта правильного функционирования системы;

1.3 Выбор архитектуры

Было рассмотрено три архитектурных паттерна – MVC, MVP, MVVM.

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Режим MVC является одним из самых классических режимов разработки, который разделен на три части модель, вид, контроллер.

Модель (Model) - представляет данные Моделью может быть один объект или некоторая структура объектов.

Представление (View) является визуальным представлением своей модели. Обычно оно выделяет одни атрибуты модели и подавляет другие. Таким образом, представление действует как фильтр представления.

Контроллер (Controller) является связующим звеном между пользователем и системой. Он предоставляет пользователю входные данные, организуя отображение соответствующих представлений в соответствующих местах на экране. Он предоставляет средства для пользовательского вывода, предоставляя пользователю меню или другие средства предоставления команд и данных. Контроллер получает такой пользовательский вывод, переводит его в соответствующие сообщения и передает эти сообщения одному или нескольким представлениям. Контроллер не обновляет представление напрямую.

Архитектура MVC для данного приложения не подходит, так как:

- приложение на этой архитектуре сложно тестировать;
- сложно реализовать в чистом виде;
- непонятно что отвечает за UI
- код контроллера разрастается

На рисунке 1 можно увидеть схему архитектуры MVC.

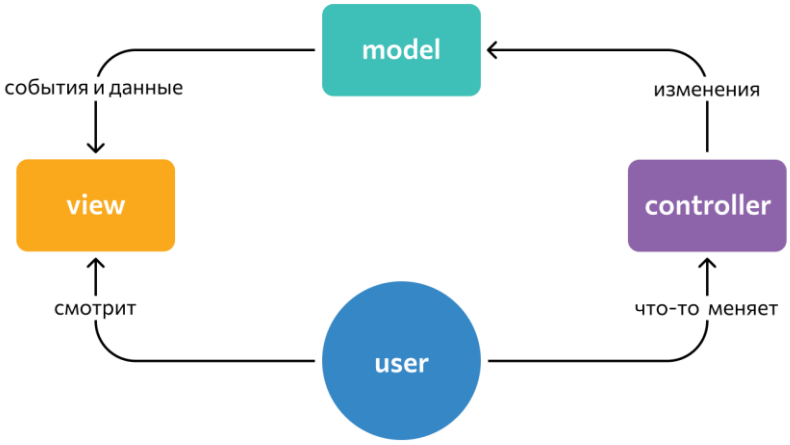

Рисунок 1 Схема MVC

В итоге что бы реализовать чистый MVC необходимо приложить большие усилий, а для маленькой команды разработчиков это критично.

Проблемы MVC хорошо решены в архитектурах MVP и MVVM. Так что стоит рассмотреть их.

Model-View-Presenter (MVP) — шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса.

Элемент Presenter в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса так же, как в других шаблонах обычно отвечает представление.

Архитектурный паттерн MVP предназначен главным образом для улучшения паттерна архитектуры MVC в Android. Наиболее отличительным моментом между MVP и MVC является то, что нет прямой связи между Model и View. Между ними существует разрыв. Это уровень Presenter, который отвечает за регулирование косвенного взаимодействия между View и Model.

Model/View/ViewModel — это вариант модели/представления/контроллера (MVC), адаптированный для современных платформ разработки пользовательского интерфейса, где представление является обязанностью дизайнера, а не классического разработчика. Шаблон архитектуры MVVM в основном нацелен на улучшение шаблонов архитектуры переднего плана и архитектуры MVC, снижение нагрузки на уровень контроллера или уровень представления и достижение более четкого кода. Благодаря инкапсуляции уровня ViewModel: инкапсуляция обработки бизнес-логики, инкапсуляция сетевой обработки, инкапсуляция кэширования данных и т. д. логическая обработка разделена, и нет необходимости обрабатывать данные модели, делая структуру уровня контроллера или уровня слоя простой и понятной.

Архитектуры MVP и MVVM отлично решают проблемы с тестированием и разделяют Вид и Модель, все компоненты растут сбалансировано. Рисунок 2 и рисунок 3 показывают схемы работы архитектурных паттернов MVP и MVVM соответственно. В отличие от MVC, в MVVM бизнес-логику не всегда легко переиспользовать, а ViewModel может разрастаться. Архитектура MVVM подходит для сложных проектов с средней или большой командой разработчиков.

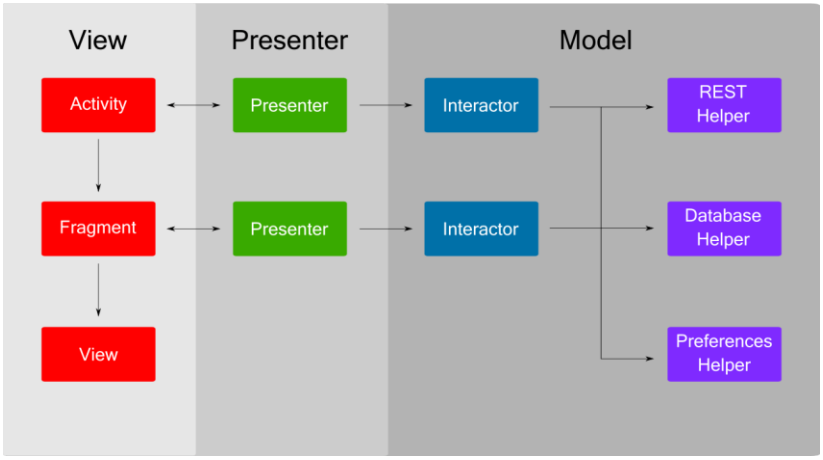

Рисунок 2 Схема MVP

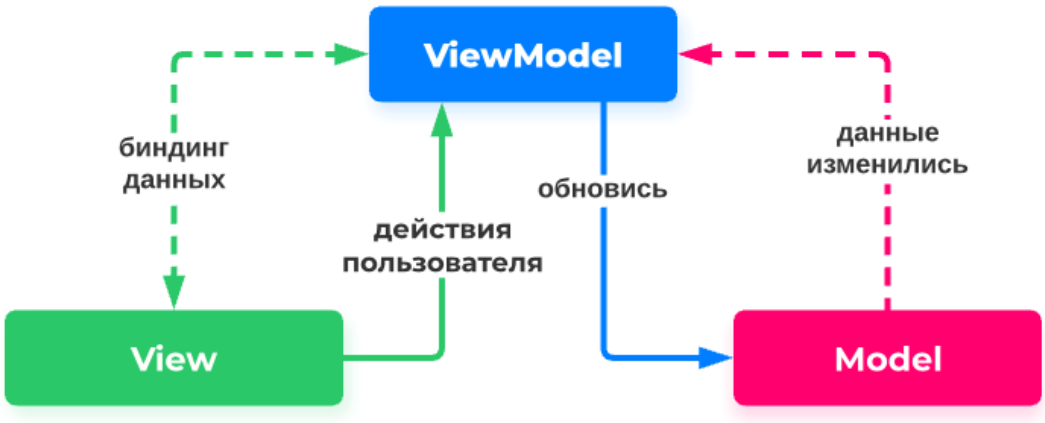

Рисунок 3 Схема MVVM

В архитектуре MVP нет таких проблем, он прост в использовании и в исполнении, подходит для маленьких команд разработчиков и подходит для

небольших приложений. Эту архитектуру легко тестировать, все разделено естественным образом и все компоненты растут сбалансированно.

Исходя из всего этого архитектура MVP отлично подходит для данного приложения.

2. Проектирование архитектуры

2.1Моделирование архитектуры

В данной работе мной была выбрана архитектура MVP.

Model View Presenter (MVP) — это новейший и лучший шаблон архитектуры Android, который помогает отделить бизнес-логику (модель) от логики представления (активность/фрагмент) с помощью промежуточного шага, который называется Presenter. Схема MVP на Рисунке 2. Данная архитектура разделена на три части – Model, View, Presenter. Проектирование я начну с модели.

Для начала необходимо выделить сущности. Я выделила четыре сущности, которые можно увидеть на рисунке 4:

- Users – информация о пользователе, логин и номер телефона.
- PlaceMark – информация о метке на карте, местоположение, автор метки, время создания/удаления.
- Messages – сообщения пользователя другому пользователю, текст сообщения, логин пользователя, время отправки.
- Chats – чат двух пользователей, номер телефона пользователя, логин пользователя кому отправили, список сообщений между этими пользователями.

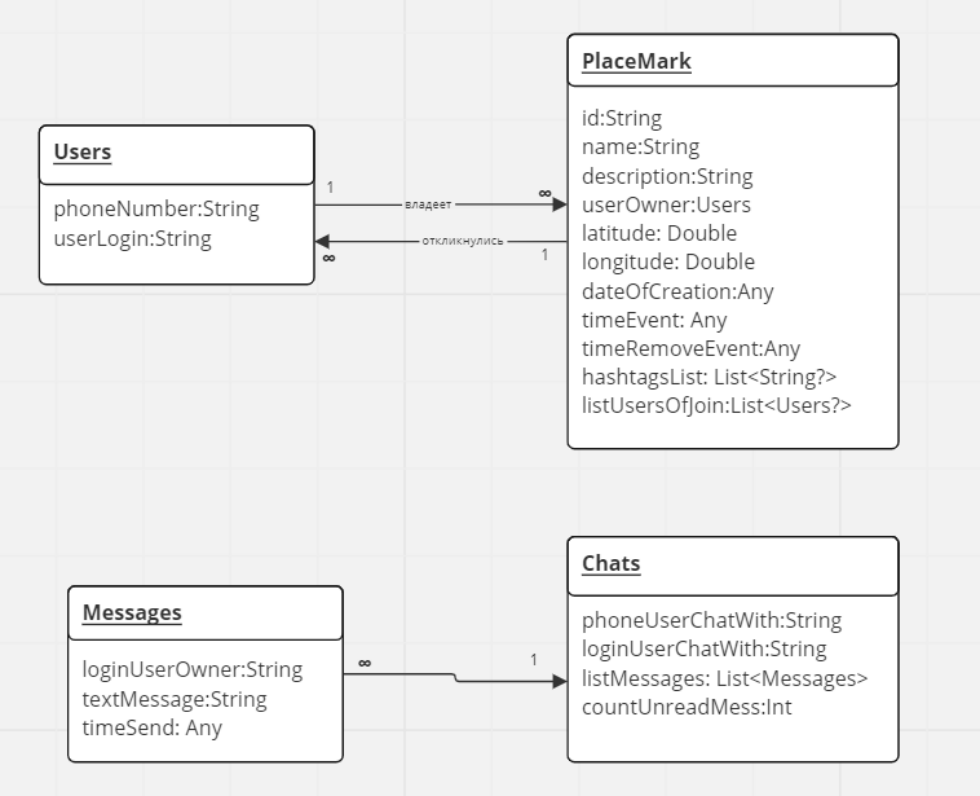

Рисунок 4 Сущности

Как видно на Рисунке 4, проект можно разделить на две части. Первая будет завязана на пользователях(Users) и метках (PlaceMark), вторая будет связана на функции мессенджера в приложении (Chats и Message).

Сущности, которые были описаны выше – это по сути классы Model в архитектуре MVP. У них нет никаких методов и функций, за исключением методов get() и/или set(). Это каркас для данных, которые приходят с БД. Поэтому можно переходить к Presenter, который и будет производить всю бизнес-логику в данном приложении. На Рисунке 6 показано какие функции будет делать Presenter в модуле работы с картой и метками приложения. Их будет несколько для разных функций. У каждого Presenter, есть интерфейсы для переиспользования. На рисунке 5 можно увидеть Presenter для модуля работы с картой и метками в приложении.

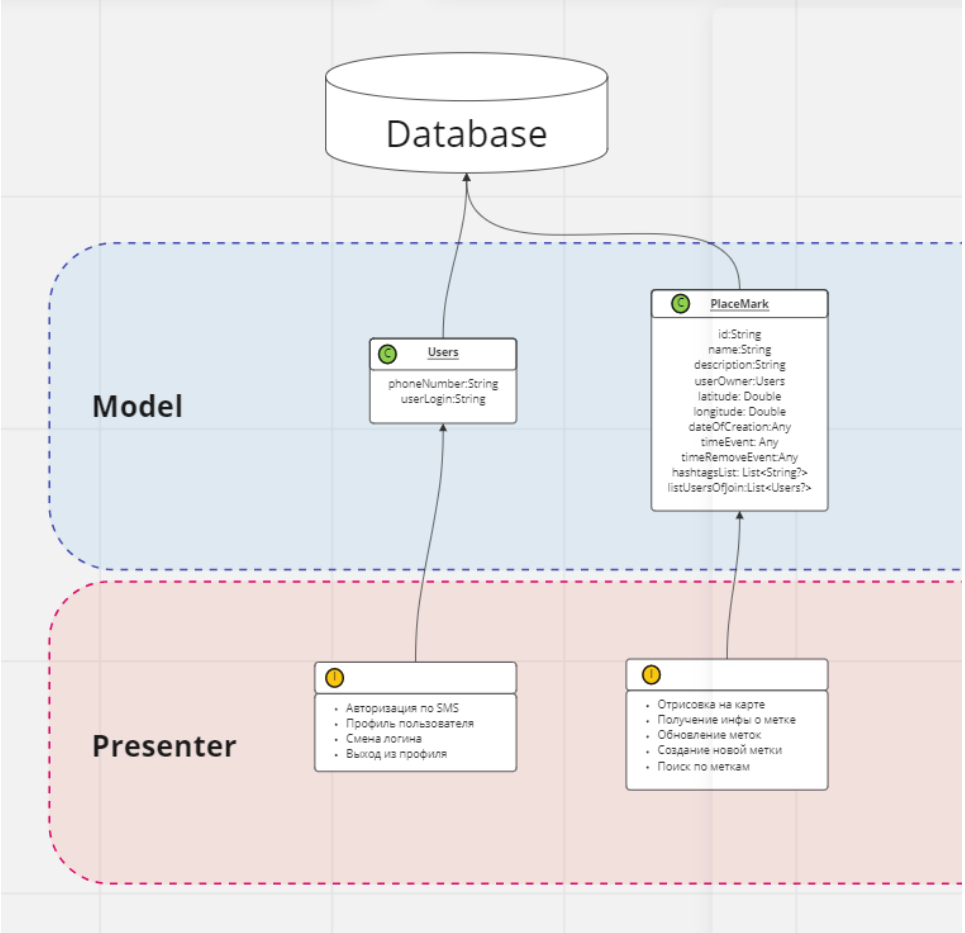

Рисунок 5

Далее переходим к View. На Рисунке 6 можно увидеть полную схему модерирования архитектуры в приложении.

Необходимые View в приложении:

- MainActivity – стартовый экран при входе в приложение, на нем будет карта с метками, кнопки для перехода в профиль пользователя, кнопка для добавления новой метки и кнопка для перехода к сообщениям пользователя.
- ProfilCurrentUserActivity – открывается по кнопке для перехода к профилю пользователя, будет загружать фрагмент LogInFragment, если пользователь не авторизован в приложении или фрагмент ProfilUserFragment, если пользователь авторизован.

- AddPlaceMarkActivity – открывается по тапу на карте или по нажатию кнопки добавить метку на карту. В этой активити нужно будет ввести информацию о новой метке, которую хочет создать пользователь.
- ChatsActivity – экран чатов текущего пользователя. Открывается с главного экрана по нажатию кнопки «Сообщения» или по нажатию логина владельца метки с диалогового окна с информацией о метке, а также при нажатии кнопки «Я пойду».
- MessagesActivity – экран чата с конкретным пользователем. На этом экране происходит общение двух пользователей с помощью отправки сообщений.

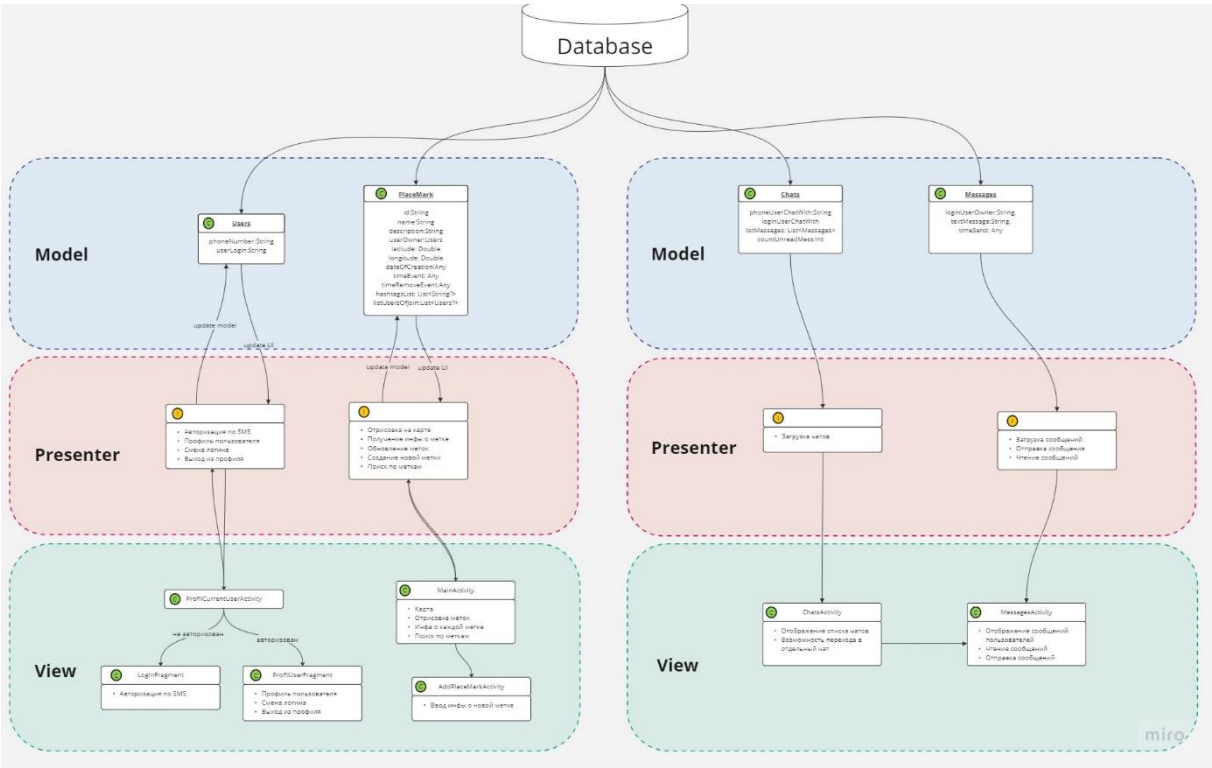

Рисунок 6 Схема моделирования архитектуры в приложении

На этом моделирование архитектуры законченно, поэтому можно переходить к подбору технологий для реализации.

3. Подбор технологий для реализации

3.1 Выбор языка разработки

Невозможно предложить универсальный набор технологий для всех. Каждый проект уникален и имеет свои требования.

Вместо того чтобы слепо копировать технологический стек, используемый другими приложениями, лучше разобраться, что подойдет данному проекту.

Выбор технического стека должен отражать опыт команды. Будет бессмысленно выбирать инструменты, которыми не владеют выбранные разработчики.

Для начала необходимо выбрать язык программирования. В 2022 году для нативной разработки Android приложений на выбор есть два языка программирования – Kotlin или Java.

Язык Java создали в компании Sun Microsystems в начале 90-х годов прошлого столетия. Этот язык появился в ответ на потребность в платформенной независимости. Java стал языком общего назначения, на котором можно писать программы под любые устройства: ПК, бытовую технику и мобильные устройства. Эта особенность языка, наряду с простотой синтаксиса и сильной типизацией почти сразу покорила сердца множества программистов. Сегодня большинство корпоративных проектов и веб-сайтов пишутся не без помощи Java, а аудитория языка насчитывает более 10 млн программистов.

Kotlin — это ещё совсем молодой язык, родившийся в российской компании JetBrains в 2011 году. Как Java, C++ и C, Kotlin — статически типизированный язык. Это позволяет ему проверять типы данных в программе во время компиляции, и своевременно выявлять ошибки. То есть если компилятор замечает нарушение правил языка, то программа просто не

выполняется. Но главная особенность Kotlin в том, что язык полностью совместим с Java, что позволяет не переписывать имеющиеся приложения. Именно это и покорило программистов в Google.

И Kotlin, и Java используются в разработке Android приложений в настоящий момент.

Чтобы разобраться, чем Kotlin отличается от Java, сравним их по нескольким критериям: возраст языка, количество кода, проверяемые исключения, безопасность, среда разработки и цели разработки.

Ниже можно увидеть таблицу сравнения данных языков по разным критериям. Благодаря таблице можно наглядно продемонстрировать различия языков.

Таблица 1. Различия Kotlin и Java

Критерии	Kotlin	Java
Возраст языка	Kotlin появился в 2011 году, и молодость языка в этом случае не играет ему на руку. Чтобы найти ответ на какой-то вопрос или решение задачи, придётся потратить немало времени. Даже документация здесь не сильно поможет. В большинстве случаев она сводится к тому, что задачи решаются	Язык появился в 1995 году. За это время вокруг него сформировалось большое комьюнити, и это позволяет без проблем найти ответ на вопрос, даже очень специфичный. Вы можете почитать документацию, посмотреть видеоролик или обратиться к другим разработчикам — с

	похоже с Java, но с небольшими изменениями	вероятностью 99% ответ будет найден
Количество кода	Чтобы написать на Kotlin такую же программу, как на Java, у вас уйдёт меньше времени и строк кода. Это очень удобно, когда требуется создать проект в короткие сроки	Особенности синтаксиса Java предполагают, что код будет более громоздким, чем в Kotlin. Вдобавок язык крайне консервативен, поэтому важно следить за мелкими деталями
Проверяемые исключения	В языке не предусмотрены условия для проверяемых исключений, а значит нет необходимости объявлять какие-либо исключения. Код сокращается — безопасность повышается	Проверяемые исключения — головная боль Java-программистов. Компилятор заставляет каждый привлекающий внимание метод обозначать как исключение. Разработчик вынужден обрабатывать его, а это сказывается на скорости разработки
Безопасность	В Kotlin по умолчанию встроена null-безопасность поэтому если программист	Ещё одна проблема Java-разработчиков связана с так называемой «нулевой безопасностью» или

	решит присвоить чему-то значение null — во время компиляции просто произойдёт сбой. Для разработчиков null-безопасность языка позволяет не писать дополнительный код, что является очередным преимуществом языка перед Java	исключением NullPointerExceptions. В Android используется null для представления отсутствия значений, но эта же ситуация может попросту уничтожить программу
Среда разработки	Android Studio базируется на программе Intellij Idea, которая создана и поддерживается той же компанией, что и Kotlin — Jetbrains. Поэтому Kotlin-разработчики получают актуальную IDE, которая покрывает все их запросы	При разработке приложений под Android в Java используется специализированная среда разработки IDE — Android Studio. Она заточена под работу с Java, поэтому вы можете начинать писать строку, а IDE сама её закончит
Цели разработки	Если вы хотите создать приложение с долгим сроком жизни, то оптимальным вариантом будет выбор	Большинство приложения для Android написаны на Java, и маловероятно, что кто-то решит переписать их на

	Kotlin. Особенно если учитывать простоту разработки на нём и множество фишек, которых нет в Java	Kotlin. Также, когда дело касается инновационных решений, VR/AR-приложений или робомедицины

Kotlin показывает себя лучше в быстроте и простоте написания кода, он более современный и нестрогий. Java, наоборот, — более строгий и консервативный язык, но открывает для разработчика возможности за пределами мобильной разработки. С 2019 года компания Google называет его основным языком android-разработки. Так как для данного мобильного приложения нам не нужно ничего кроме разработки самого приложения, Kotlin будет лучшим выбором.

3.2 IDE

Интегрированная среда разработки (IDE) – это программное приложение, которое помогает программистам эффективно разрабатывать программный код. Оно повышает производительность разработчиков, объединяя такие возможности, как редактирование, создание, тестирование и упаковка программного обеспечения в простом для использования приложении.

Для написания кода можно использовать любой текстовый редактор. Однако большинство интегрированных сред разработки (IDE) включают в себя функции, выходящие за рамки редактирования текста. Они предоставляют центральный интерфейс для общих инструментов разработчика, делая процесс разработки программного обеспечения гораздо более эффективным. Разработчики могут быстро приступить к программированию новых приложений вместо того, чтобы вручную интегрировать и настраивать различное программное обеспечение. Кроме

того, им не нужно изучать все инструменты, а можно сосредоточиться только на одном приложении.

Для разработки Android приложений существует отличная IDE - Android Studio.

Android Studio — интегрированная среда разработки производства Google, с помощью которой разработчикам становятся доступны инструменты для создания приложений на платформе Android OS. Android Studio можно установить на Windows, Mac и Linux. IDE можно загрузить и пользоваться бесплатно. В ней присутствуют макеты для создания UI, с чего обычно начинается работа над приложением. В Studio содержатся инструменты для разработки решений для смартфонов и планшетов, а также новые технологические решения для Android TV, Android Wear, Android Auto, Glass и дополнительные контекстуальные модули.

Android Studio — официальная интегрированная среда разработки (IDE). для разработки приложений для андроид, основанный на программном обеспечении JetBrains IntelliJ IDEA. Для поддержки разработки приложений в операционной системе Android Android Studio использует систему сборки на основе Gradle, эмуляторы, шаблоны кода и интеграцию с Github. Android Studio — это официальная интегрированная среда разработки (IDE) Google для операционной системы Android, построенная на основе программного обеспечения JetBrains IntelliJ IDEA и разработанная специально для разработки под Android. Система сборки Android — это набор инструментов, используемых для создания, тестирования, запуска и упаковки ваших приложений.

Требования и поддерживаемые операционные системы:

- процессор x86_64; Intel Core 2-го поколения или выше или аналог AMD.

- 8 ГБ оперативной памяти или больше.
- 8 ГБ свободного места на жестком диске (IDE + Android SDK + эмулятор Android)
- Экран с минимальным разрешением 1280x800.

Одна из самых интересных частей Android Studio, которая заставляет многих разработчиков устанавливать это программное обеспечение на свои компьютеры для запуска собственных приложений Android, — это его эмулятор. Эмулятор Android Studio интегрированный позволит вам легко тестировать все виды нативных приложений, как если бы у вас было мобильное устройство. Кроме того, они поддерживают возможность выбора между различными версиями Android для тестирования в разных средах, а также между различными популярными мобильными устройствами на рынке, такими как Google Pixel, Samsung Galaxy и т. д.

Поэтому Android Studio – отличный выбор для разработки данного мобильно приложения.

3.2 Пользовательский интерфейс приложения

В разработке мобильного приложения для организации встреч и мероприятий делается упор на техническую часть, а не на визуальную составляющую. Поэтому для дизайна данного приложения не будут использоваться дополнительные фрейморки или другие технические средства. Но благодаря хорошо подобранной архитектуре приложения, не составит труда добавить в команду дизайнера и спроектировать хороший дизайн приложения.

Для дизайна приложения, будут использоваться стандартные ресурсы Android Studio, а именно разметка XML. Библиотеки Android имеют набор классов для работы с XML-документами с произвольной структурой и содержанием.

Расширяемый язык разметки (XML) – это язык разметки, который предоставляет правила для определения любых данных. В отличие от языков программирования, язык разметки XML не может выполнять вычислительные операции сам по себе. Вместо этого для управления структурированными данными можно использовать любой язык программирования или программное обеспечение. И этого будет вполне достаточно для дизайна данного приложения.

3.3Выбор базы данных

Разработка надежных и высококачественных приложений для мобильных устройств, предполагает огромную самоотдачу, но что еще более важно, требует мощную и многофункциональную платформу для разработки.

Firebase, предоставляемая компанией Google, является одной из таких платформ, которая завоевала прочные позиции среди разработчиков по всему миру.

Firebase предоставляет разработчикам множество возможностей для создания высоко эффективных и универсальных веб-приложений, а также приложений для платформ Android и iOS.

Firebase- это платформа для разработки мобильных приложений от компании Google, в которой есть самые современные функции для разработки, перекомпоновки и улучшения приложений.

Firebase включает две базы данных, это Cloud Firestore и Realtime Database, которые являются полезными инструментами, удовлетворяющие современным требованиям разработки приложений. Для разработки данного приложения была выбрана Realtime Database. [7]

База Данных Firebase Realtime это облачная база данных. Она облегчает хранение данных на основе JSON и выполняет синхронизацию данных в реальном времени с подключенными клиентами. Отдельные экземпляры базы данных функционируют как клиенты в процессе разработки кроссплатформенных приложений с использованием SDKiOS, JavaScript и Android. Это позволяет приложениям получать обновления и данные в реальном времени. В автономных приложениях данные никуда не теряются, поскольку SDK базы данных обеспечивает сохранение данных на диске. Это помогает синхронизировать устройства с серверами после восстановления подключения.

Поэтому база данных Firebase Realtime это отличный выбор в данном случае.

Отмечу, что при ограничении доступа Firebase Realtime, благодаря хорошо спроектированной архитектуре, приложение можно перевести на другие системы бэкенда и подключить другую базу данных, с минимальными усилиями от разработчиков.

3.4 Выбор API для отображения карты

В мобильном Android-приложении для организации мероприятий и встреч должна присутствовать карта, для того что бы отобразить метки пользователей по заданным координатам. Для этого было выбрано API Yandex MapKit.

MapKit — это программная библиотека, которая позволяет использовать картографические данные и технологии Яндекса в мобильных приложениях. [6]

MapKit позволяет создать приложение с картами Яндекса для операционных систем iOS и Android. В таком приложении можно использовать поиск по организациям и топонимам, строить автомобильные

и пешеходные маршруты с учётом актуальной дорожной ситуации, а также отображать информацию о пробках или панорамы.

На созданной с помощью MapKit карте будут появляться строящиеся дома, новые дороги и развязки, недавно открытые организации. Словом, на ней будут доступны все те изменения, которые вносятся на Яндекс Карты.

У API Yandex MapKit отличная и понятная документация, что позволяет без проблем использовать это в своем проекте.

4. Реализация приложения

4.1Реализация профиля пользователя

Так как была выбрана архитектура MVP, разработка профиля пользователя был создан класс модели, его можно увидеть на рисунок 7. Это дата класс Users, который хранит в себе поля phoneNumber (номер телефона пользователя), userLogin (логин пользователя) и listPlaceMarkOfJoin (список id меток на которые пользователь планирует пойти).

```
data class Users(
    val phoneNumber:String? = null,
    val userLogin:String? = null,
    var listPlaceMarkOfJoin: List<String> = listOf()
){
}
```

Рисунок 7 Класс Users

Вся бизнес-логика должна находиться в Presenter. С помощью интерфейсов Presenter передает результат в View. View и Model никак между собой не контактируют. Например, для авторизации пользователя будет такая последовательность действий:

1. Пользователь вводит номер телефона и нажимает кнопку «Получить код», в коде мы, после события onClick(), считываем номер телефона пользователя и отправляем в соответствующий Presenter
2. После чего Presenter проверяет что поле не пустое и соответствует формату и делает запрос в Firebase.
3. Firebase отправляет смс с кодом для авторизации на телефон пользователя
4. Открывается View для ввода кода с смс
5. Пользователь вводит код и нажимает кнопку «Отправить»

6. После события onClick(), отправляем в Presenter введённый код.
7. Presenter отправляет его Firebase и ловит ответ
8. Если код правильный Presenter через интерфейс сообщает View, что все корректно, если код неправильный Presenter через другой интерфейс сообщает View об ошибке

Элемент Presenter в шаблоне MVP берёт на себя функциональность посредника и отвечает за управление событиями пользовательского интерфейса так же, как в других шаблонах обычно отвечает представление. У каждого Presenter есть один или несколько интерфейсов для взаимодействия с View.

Далее был реализован Presenter. Он реализован с помощью классов и интерфейсов. Каждый класс переопределяет соответствующий интерфейс и отвечает за свою задачу. На рисунке 8 можно увидеть список представлений в проекте для данного модуля.

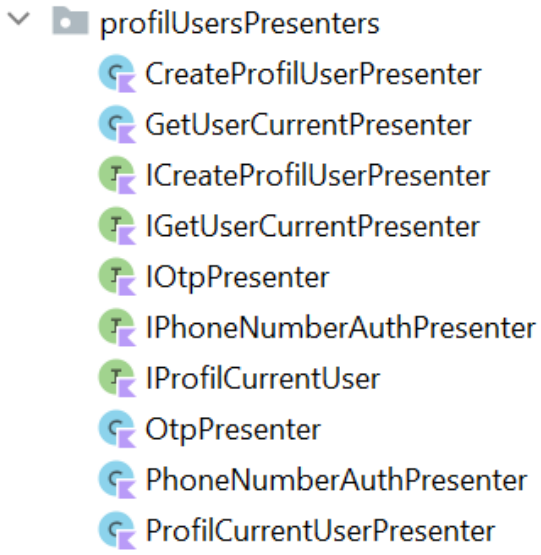

Рисунок 8 Структура Presenter для профиля пользователя

Также из каждый Presenter через интерфейс передает данные в View для последующих изменений, которые увидит пользователь. View-интерфейсы

имплементируются в классах активити и фрагментах. И функции из них переопределяются в зависимости от того что нам нужно в конкретном активити или фрагменте.

Для авторизации пользователей была использована аутентификация по номеру телефона от Firebase. Пользователю нужно ввести номер телефона и получить смс с кодом. Далее он вводится и, если код правильный аутентификация проходит успешно. После чего пользователь может в полной мере пользоваться приложением.

Были разработана следующая View - ProfilCurrentUserActivity – открывается по кнопке для перехода к профилю пользователя, будет загружать фрагменты:

- LogInFragment, если пользователь не авторизован в приложении или фрагмент. Вид фрагмента на рисунке 9;

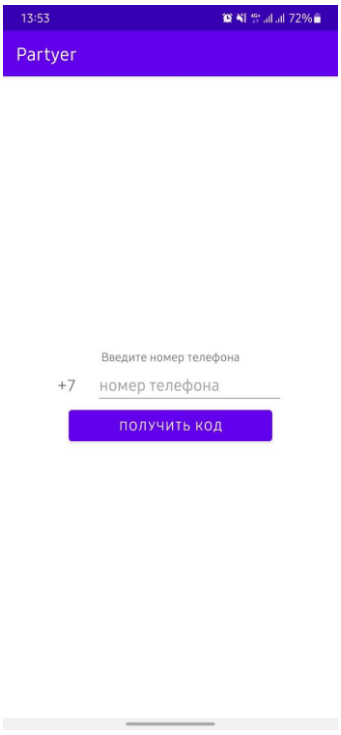

Рисунок 9 LogInFragment

- ProfilUserFragment, если пользователь авторизован. Вид фрагмента на рисунке 10.

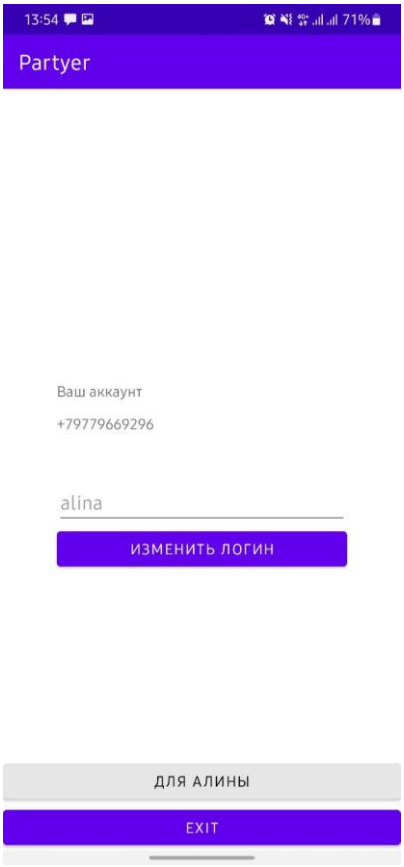

Рисунок 10 ProfilUserFragment

4.2Реализация работы с метками

Для реализации работы с метками необходимо подключить Yandex MapKit API. Это делается с помощью подключения библиотеки.

```
implementation 'com.yandex.android:maps.mobile:4.3.1-full'
```

Также был создан класс модели, можно увидеть на рисунке 11, в котором есть все необходимые поля для отрисовки меток и получения информации о метке на карте.

```
data class PlaceMark(
    var id:String? = null,
    val name:String? = null,
    val description:String? = null,
    val userOwner:Users? = null,
    val latitude: Double? = null,
    val longitude: Double? = null,
    val dateOfCreation:Any? = null,
    var timeEvent: Any? = null,
    var timeRemoveEvent:Any? = null,
    val hashtagsList: List<String?> = emptyList(),
    val listUsersOfJoin: List<String> = listOf("hi"),
) {
```

Рисунок 11 Класс PlaceMark

Также, как и в случае с профилем пользователя, были созданы Presenter. Их список в проекте на рисунке 12.

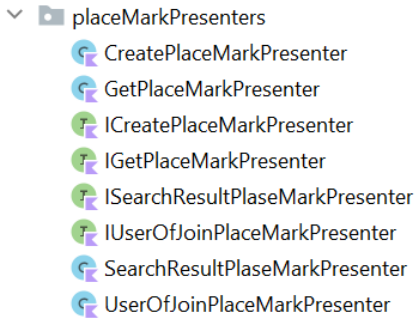

Рисунок 12 Структура Presenter для меток

Вся информация хранится в базе данных Firebase Realtime Database Как данные хранятся в базе данных можно увидеть на рисунке 13.

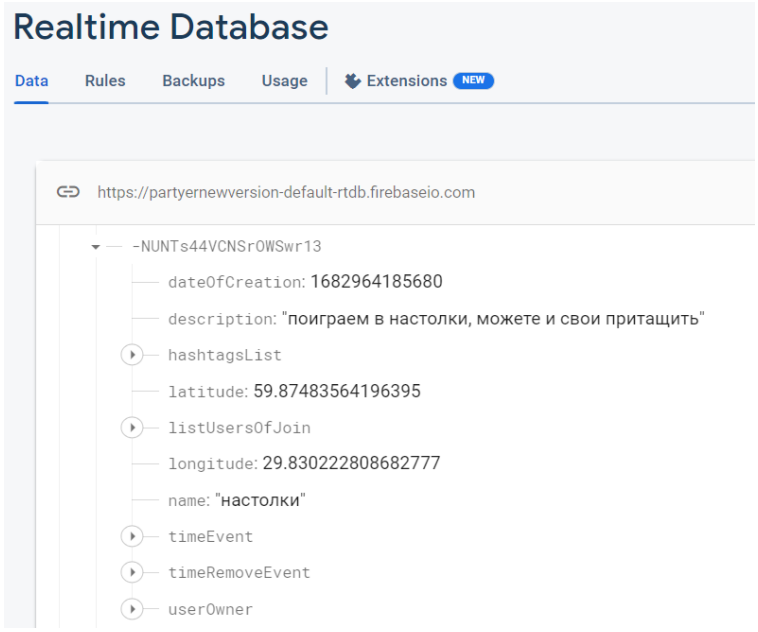

Рисунок 13 Хранение информации о метке в Firebase

При запуске приложения, происходит загрузка меток из БД в Presenter рисунок 14. При получении метки из БД, она сериализуется в соответствующий объект класса, и этот объект передается в View для отрисовке на карте.

```
AGudzenko *
override fun getAllPlaceMarks() {
    ref.child( pathString: "PlaceMark").addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            for (ds in snapshot.children) {
                val mark: PlaceMark? = ds.getValue(PlaceMark::class.java)
                mark?.id = ds.key
                val timeRemoveLong = timestampToLong(mark?.timeRemoveEvent)
                val dateTimeCurrent = Date()
                val dateTimeRemove = Date( date: timeRemoveLong!!.toLong() * 1000)
                mark?.timeEvent = timestampToDateString(timestampToLong(mark?.timeEvent)!!)
                mark?.timeRemoveEvent = timestampToDateString(timestampToLong(mark?.timeRemoveEvent
                if (dateTimeCurrent.time < dateTimeRemove.time) {
                    getPlaceMarkView.showPlaceMark(mark)
                } else {
                    ds.ref.setValue(null)
                }
            }
        }

        override fun onCancelled(error: DatabaseError) {
            Log.w( tag: "ERROR",  msg: "Failed to read value.", error.toException())
            getPlaceMarkView.errorGetPlaceMark("Failed to read value." + error.toException())
        }
```

Активация Windows
Чтобы активировать Windows, перейдите в

Рисунок 14 Получение меток из БД

При запуске приложения пользователь видит карту с метками, все метки делятся на три группы. Внешний вид меток можно увидеть на рисунке 15. Первая «пустая» - метки других пользователей на которые пользователь не откликнулся, вторая «звездочка» - метки, которые пользователь создал самостоятельно, третья «сердце» - метки, на которые пользователь планирует пойти.

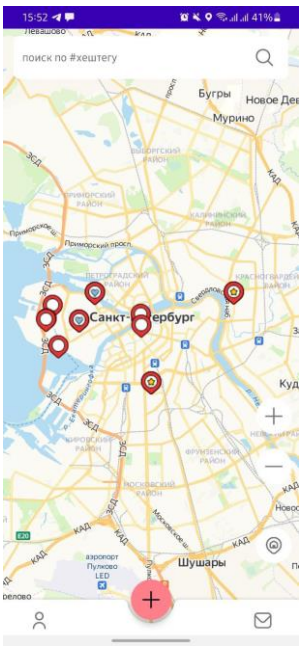

Рисунок 15 Метки на карте

Также, у пользователя есть возможность искать мероприятия по хештегам. Каждый пользователь по желанию указывает хештеги, которые описывают его мероприятие. На главном экране есть поле для поиска по хештегу. Пользователь вводит хештег и запускает поиск. Найденные метки отображаются на карте.

При нажатии на метку отображается информация про данное мероприятие, что можно увидеть на рисунке 16. Пользователь может нажать на кнопку «Я пойду» для отклика, на логин создателя метки, для связи, а также, на один из хештегов, для поиска подобных мероприятий.

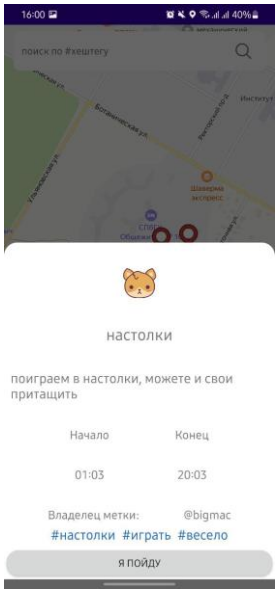

Рисунок 16 Информация о мероприятии

4. 3 Реализация мессенджера в приложении

В данном приложении реализована возможность общения между двумя пользователями – текущим пользователем и владельцем какой-либо метки. Для реализации данного функционала были созданы два класса модели Chats и Message. Их можно увидеть на рисунке 17. Данные классы представляют собой Model.

```
data class Chats(
    var phoneUserChatWith:String? = null,
    var loginUserChatWith:String? = null,
    var listMessages: List<Messages> = emptyList(),
    var countUnreadMess:Int? = 0) {
}

class Messages(
    val loginUserOwner:String? = null,
    val textMessage:String? = null,
    val timeSend: Any? = null) {
```

Рисунок 17 Классы Chats и Messages

В архитектуре MVP классы Модели не должны содержать никаких методов, кроме get() или set(). На рисунке 18 можно увидеть, как выглядит класс Messages. Ничего кроме полей класса и методов get(), set() для установки корректной даты.

```
AGudzenko
class Messages(
    val loginUserOwner:String? = null,
    val textMessage:String? = null,
    val timeSend: Any? = null) {

    var createdTimestamp:Any? = timeSend
        set(value) {
            field = ServerValue.TIMESTAMP
        }
        @Exclude
        get() {
            if(timeSend == null){
                return 0
            } else {
                return timeSend as Long
            }
        }
}
```

Рисунок 18 Класс Messages

В базе данных все хранится в виде json, и выглядит таким образом Рисунок 19.

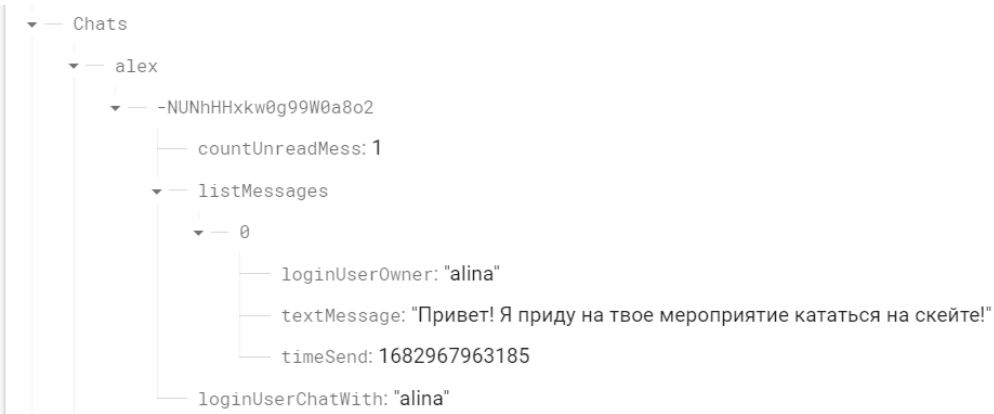

Рисунок 19 Хранение в базе данных

Далее были созданы view дляданного функционала. Созданные View в приложении:

- ChatsActivity – экран, который доступен по кнопке «сообщения» на главном экране. На этом экране пользователю отображаются все его чаты. По нажатию на один из чатов откроется MessagesActivity. Рисунок 20.

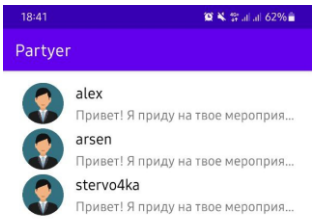

Рисунок 20 ChatsActivity

- MessagesActivity– открывается по нажатию на чат в экране чатов, а также по нажатию на логин владельца метки на окне с информацией о мероприятии или по нажатию кнопки «Я пойду», после чего автоматически отправляется сообщение владельцу метки, видно на рисунок 21.

Рисунок 21 MessagesActivity

Заключение

В результате выполнения работы, было разработано приложение на платформе Android для организации встреч и мероприятий.

В ходе выполнения данной работы были выполнены все поставленные задачи:

- формирование требований к мобильному приложению и выбор подходящей архитектуры
- проектирование архитектуры мобильного приложения
- подбор технологий для реализации
- реализация данного мобильного приложения

Список использованных источников

1. Стивен Вальтер, «Эволюция MVC», 24 августа 2008 г. - статья
2. Маной Джаггаварапу, «Шаблоны презентаций: MVC, MVP, PM, MVVM», 02.05.2012 - статья
3. Джефф Анджелини, «Шаблон MVP для Android», 10 апреля 2015 г. - статья
4. Ханнес Дорфманн, «Model-View-Presenter», 25 мая 2015 г. - статья
5. Джо Берч, «Подход к Android с помощью MVVM», 21 сентября 2015 г. - статья
6. yandex.ru/dev/maps/mapkit/ - документация для разработчиков
7. firebase.google.com/ - документация для разработчиков